Ahmed Mursalat

QuestionDB: A database of questions

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

28 February 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Ahmed Mursalat QuestionDB: A database of questions 47 pages 28 February 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Architecture, User Interactions, Data Handling
Instructor(s)	Jaana Holvikivi Taru Sotavalta

The aim was to develop a system that stores questions and answers in different fields of study. These questions and answers were provided by the users, through a crowd-sourced mechanism. The clearest option was a system designed as a web based solution.

In order to create such a system MongoDB can achieve a good performance in the data storage and at the same time allow for a schema-less storage of the data. Node.js was chosen to handle the application needs otherwise. An adherence to the MVC policies were partially held true to, however distinct code was used for the models. Nginx was then reverse proxying all requests from the outside world into the node application which handled them.

While the system was feature-rich in terms of rendering Latex with a drag and drop feature for images, there were multiple issues outstanding. From a completely technical perspective, there was pretty significant performance drop when there were only 30 concurrent users. But from a more ideological angle the system should support multiple types of questions. While currently only multiple-choice questions can be created. Filling in the blanks is another perhaps obvious type of question that can be maintained by the system.

Keywords	Node.js, MongoDB, Nginx

Contents

1 Introduction

In order for a student to learn any given subject it is a good idea to practice. Only then can she/he make enough mistakes and figure the reasons behind those mistakes, this understanding can allow the student to improve. QuestionsDB is an application that is developed in order to allow students to have a crowd-sourced database of questions and solutions. The aim is to develop a functional application with minimal features to allow students to practice questions and view their answers. This poses many varying issues ranging from data structure choices and user interface creation, and of course the features list which will basically decide how much work other aspects requires.

The application at hand naturally has to be implemented as a web application, due to its crowd-sourced requirement to get many questions input for use by other people. However this adds the complexity of the use of it in multiple devices, and also response times are somewhat important. Additionally scalability has always to be kept in mind should the application experience growth in the future.

2 The Web Application

2.1 Basic Requirements

The application is required to produce multiple-choice questions that are entered into the system's database by other users. This means that a database of questions needs to be built. Thus the user must be able to insert new questions, have the correct answers stored along with the solution. Multiple-choice questions specifically are chosen due to the ease at which they can be stored in the database and the ability for the practicing student to get immediate feedback based on their answers.

The user would also be able to fetch questions based on certain parameters that they would like to define. A physics student studying logarithms might prefer to understand the topic by having many practice questions available to him. In such a situation "loga-

rithms” and “physics” are both keyword parameters that he can use to search in order to have the questions narrowed down for him.

Also in order to make the product minimally viable as a system to be used for education, the system requires easy creation and editing of questions and the solutions provided as required.

2.2 Environments

Any application requires different environments for working with. Software goes throw many life cycles, for developers it comes down to DTAP (development, testing, acceptance and production. [1])

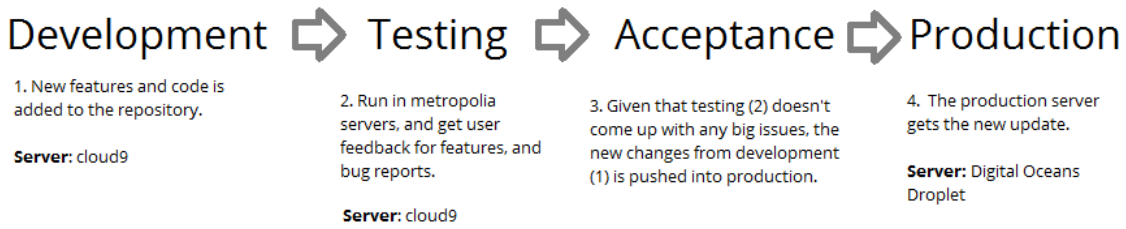


Figure 1 The different phases of a software development life cycle.

DTAP is an acronym used to describe the phases between testing the application and deployment for actual use. This is mainly for large software projects to maintain quality of the source code, and also apply the divide and conquer strategy for maintaining an active code base. An explanation and a graphical view of the phases can be found in Figure 1.

2.2.1 Version Control and Git

To maintain the code across many platforms and also the changes across them, I have used git. Git is a version control system that basically stores the differences between changes in the code over time (commonly known as diff files) and keep track of other meta data necessary for maintaining code bases (figure 2 shows the help text for the git executable file). Git's source code management abilities also makes it useful for transferring the code along the DTAP phases, thanks to its compatibility with the existing standards, such as Hyper Text Transfer Protocol (Secure - HTTPS) and Secure Shell (SSH), it also supports plain sockets for communicating over. On top of compatibility for communicating there is also git-svn, which emulates Apache Subversion (SVN) in order to provide support for existing SVN repositories and SVN users [2]. Git can informally also be used to maintain backups of the source code.

```
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty Git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Figure 2 Git console response.

The Git system is designed in a manner that is distributed since it is assumed that multiple developers will work concurrently in order to develop a code base, and thus a local

copy of the history of changes to the code is stored by Git on each developer's computer. This also means that non-linear development is emphasized by Git's management capabilities, meaning that it has a very strong support for branching and merging of code. One more point worth mentioning is that Git commits (a version of the source) contains a hash ID calculated from the development changes leading to that commit as shown in Figure 3.

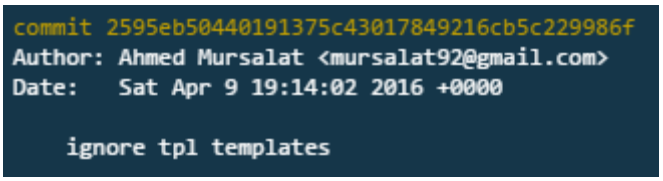


Figure 3 A Git commit record.

All source code along with revision change data is also stored in remote, which is a git term for a repository stored in a central system from where all developers of the current code base can get the latest changes to the system. Changes in the local system are first staged by first getting changes (Git pull) and merging them (Git merge). Then the new code is added and committed to the file system. The change history can then be pushed to remote, which can then be pulled by other developers. The process is repeated by others in order to maintain the code by multiple developers.

2.2.2 Development

A development environment is the platform on which the system will be deployed and ran and tested. The phased deployment allows me to test active and new changes to the system and make deployment decisions from it. Meaning whether to rollback changes for further development should an issue be detected, or to go forward with the changes.

Development is carried out in cloud9, which offers a virtual shared environment for development. Cloud9 systems also offer a web based IDE for coding [3]. This allows for rapid development and testing of changes. Due to the scale of the project I am doing,

and the reality of the availability of resources to me the staging environment will be skipped between the development and production environment.

2.2.3 Production

The main purpose of the production system is to be able to handle as many consecutive users as possible, and delivering responses on time to the end-user. It should also be able to handle most errors gracefully.

In order to do so I have configured Nginx to be the reverse proxy which takes the requests from the outside world and passes them on to the node module I have developed. The node module in turn responds to a local port that the Nginx server uses to proxy over requests from the worldwide web.

Using nginx also adds the benefit for me to run other applications in other environments. For instance I have a Django (a Python web framework for rapid web development) application running on localhost:8282 . Nginx is configured to forwards requests for foodjinx.com to localhost:8282 while the requests for qdb.mursalat.net are forwarded to localhost:3131 (as that is where QuestionDB application is listening and working on.)

3 User Interface

The user interface (UI) is the system that the end-user will use in order to view and interact with the system – it is a way for a human to interact with a machine. As a website this mainly comprises the web UI created with JavaScript/HTML/CSS.

In order to make a practical user interface, there are certain guidelines that should ideally be followed [4]:

- How clear is the user interface? Can the user use it to navigate to resources that he/she needs.

- The user should also be familiar with the UI, despite whether he/she is a first-time user or not. Parts of the UI's aspects should ideally represent real life things, such as buttons or tabs.
- The UI must be rational and persistent throughout the system so the end-user does not have to consistently learn about how to use the application. This ensures a seamless experience for the user.
- The UI should not be clot up by putting too many texts/images in a single UI. This will only make it confusing for the end-user. Putting too many elements adds the possibility of the user confusing one text for one button which it was not meant for. Although explaining the UI is a good idea - one must be careful not to make a clutter off of it in order to ensure that the user is presented with the minimal number of information. The screen that the user is using is a website's "real estate" and thus must be used effectively.

3.1 Tags

A question can be of many different types. It can be based on subject of study and depth of learning. It can also be based on certificates or merits that a user wants to achieve. In order to maximize the possibilities for end-users to classify their questions correctly into the appropriate categories (since they will be the ones who will use it in order to navigate the questions) I have decided to implement a tag based system. Whereby each category that a question falls in can be tagged with a word representing said category. The tags are then stored as embedded documents in the models (a more in depth discussion on this can be found in section 4.3.5 under models). This choice allows me to create many relations between the tags, and use the tags as the main way to search and find questions for the end-users.

3.2 WYSIWYG Editor

Most users editing documents are familiar with common short-cuts such as ctrl+b on a selected text turns it bold or ctrl+u which underlines the selected text. Since most of my

users will probably not have practical knowledge of HTML – but familiar with text editing in Microsoft word or other similar text editing packages, it is not reasonable nor user friendly to ask them to enter what they want in html. Thus I have introduced CKEditor, which is a web based editor that generates the necessary html, but the user gets to see “what you see is what you get” format of their questions (See figure 4), answers and solutions.

3.2.1 List of CKEditor Plugins

CKEditor provides a well-developed process of having plugins added into the editor, this allows for the creation of a much richer usability experience, some plugins worth mentioning follows.

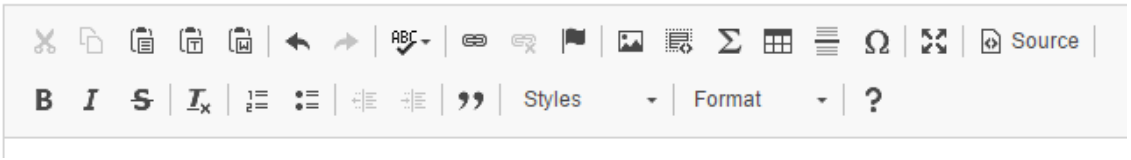


Figure 4 CKEditor menu items for wysiwyg editing of HTML

A number of topics, physics and math being the obvious ones, require the input of many equations. For this MathJax is a perfect plugin, Figure 5 shows the text area where one can enter the equations and see a live preview of the parsed equation shown. In order to allow the user to input the equations and browser to parse it so that question practitioners can see the equations in a friendly manner. MathJax is used (a more detailed description of it can be found under section 3.4)

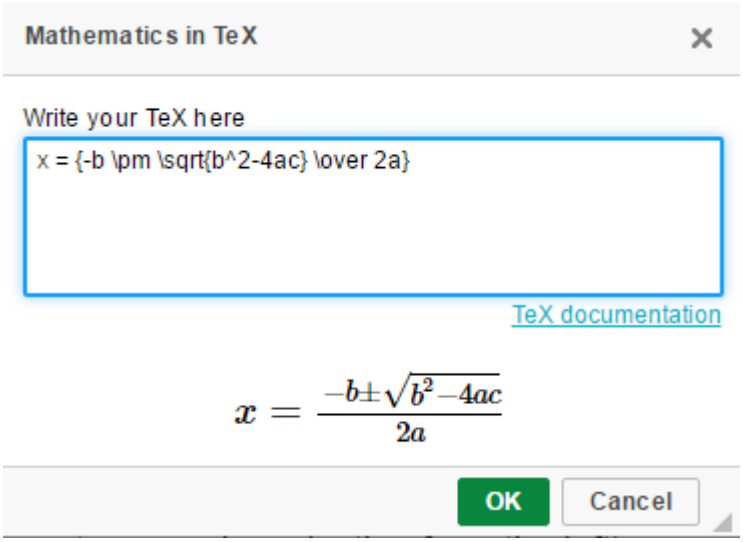


Figure 5 MathJax plugin input for formulas.

CKEditor conveniently provides a plugin that requires minimal configuration to get it up and running. The only requirement was to include the MathJax library from a CDN provider, which automatically goes through the HTML document in order to detect where to generate the proper HTML and CSS for representation of the technical texts.

While working on the project I realised that multiple-choice options can be used in order to also test knowledge of software code, and also many computer science questions rely on code. So adding code snippet was quite simple and useful for users as shown in Figure 6.

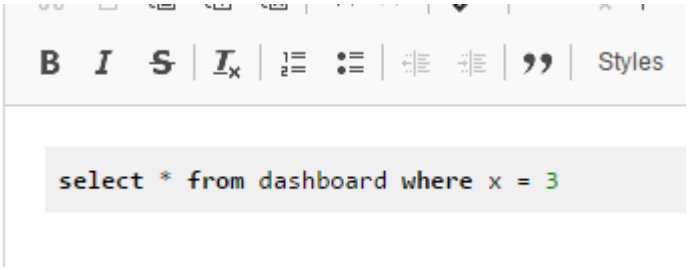


Figure 6 Code formatting in the editor.

The image upload plugin was perhaps the trickiest to implement of all. During the development phase there were a lot of errors, however the solution in the end was somewhat simpler and more elegant than I expected. I not only wanted the user to able upload images, but I wanted them to be able to drag and drop images into the docu-

ment, as most users are already familiar with this feature in software systems such as Microsoft Word Document.

Using the imageupload plugin I was able to get most of the system ready from the front-end. However CKEditor must be hooked with the backend properly with a connector in order to get this to function correctly. Since a premade connector for Node.js was not available, I improvised and made a connector of my own.

I added the url “/ckup” which only accepted POST requests. I then used middleware for node.js called multer in order to intercept and store the file.

```
327  exports.upload_file = function(req,res){
328    res.writeHead(200, {"Content-Type":"application/json"});
329    var json = JSON.stringify({
330      "uploaded": 1,
331      "filename" : req.file.originalname,
332      "url":"/images/upload/"+req.file.filename
333    });
334    res.end(json);
335  }
```

Listing 1 File upload server code.

After the file is uploaded, the multer configuration I provided, makes it change the file name, which is an md5 hash of the original file name concatenated with the timestamp. Then the code shown in Listing 1 will be ran, whereby it will send the user an “application/json” response with the success or failure of the upload process, the filename, and the URL at which it can be viewed.

3.3 MathJax

In order to understand the concept of MathJax, it is important to consider LaTeX – which can be considered as an inspiration for MathJax. LaTeX is a typesetting system that is used as the defacto standard for publishing scientific documents that require representation of equations and other technical details in projects that are medium to large. LaTeX ensures that the user is not concerned with design but more with the content of the document [5].

It is a Javascript library maintained by the American Mathematical Society that originally came from jsMath. It is a system that was developed in order to render mathematical equations into the browser via the means of using Ajax to get the image from a server side renderer, after which the browser parses it as an image. However as browsers have improved over time and computers have become more powerful. JavaScript and CSS standards have improved. They implemented more features with new abilities so that the browsers were able to render them faster and better. MathJax tries to take full advantage of these new standards and abilities, and renders equations with pure HTML/CSS/JavaScript. MathJax was started in 2004 by David Cervone. The library is provided as-is as an Apache Licensed open source software (https://www.mathjax.org/#about).

The MathJax library strives to support cross-browser (meaning that it's also cross-platform) rendering of mathematical formulas by the use of MathML, LaTeX and ASCIIMathML. It strives to provide fast and quality graphics for showing mathematical formulas and being compatible with as many browsers as possible without the need of any installation of other software or plugin.

3.4 Compatibility

All web applications have to consider compatibility issues, some more than others, mine being the latter. The front end comprises of CSS/HTML/Javascript. In the early days of web development compatibility was a very important factor for a website, but as html rendering and JavaScript engines have matured, the browser has become somewhat more forgiving than before – but less so in the case of JavaScript. Table 1 takes a look at the compatibility for the front-end technologies with the most common browsers and platforms.

Table 1. Frontend frameworks and their known compatibility.

Library	**Chrome + Firefox**	**IE**	**Opera**	**Safari**	**iOS**	**Android**
JQuery 2.1.4	Minor Issues	9+	12.1x, (Current - 1) or Current	5.1+	6.1 +	2.3, 4+

Bootstrap 2.3.2	Minor Issues	Known Issues	Not on mobile, Minor issues on desktop	Not on windows	Yes	Yes

3.5 Questions and Answers

The questions and answers are each created by a user. The user can put in a question and as many possible answers to it as possible (as shown in figure 7). This is useful if the user wants to make a question tough by giving many possible incorrect answers.

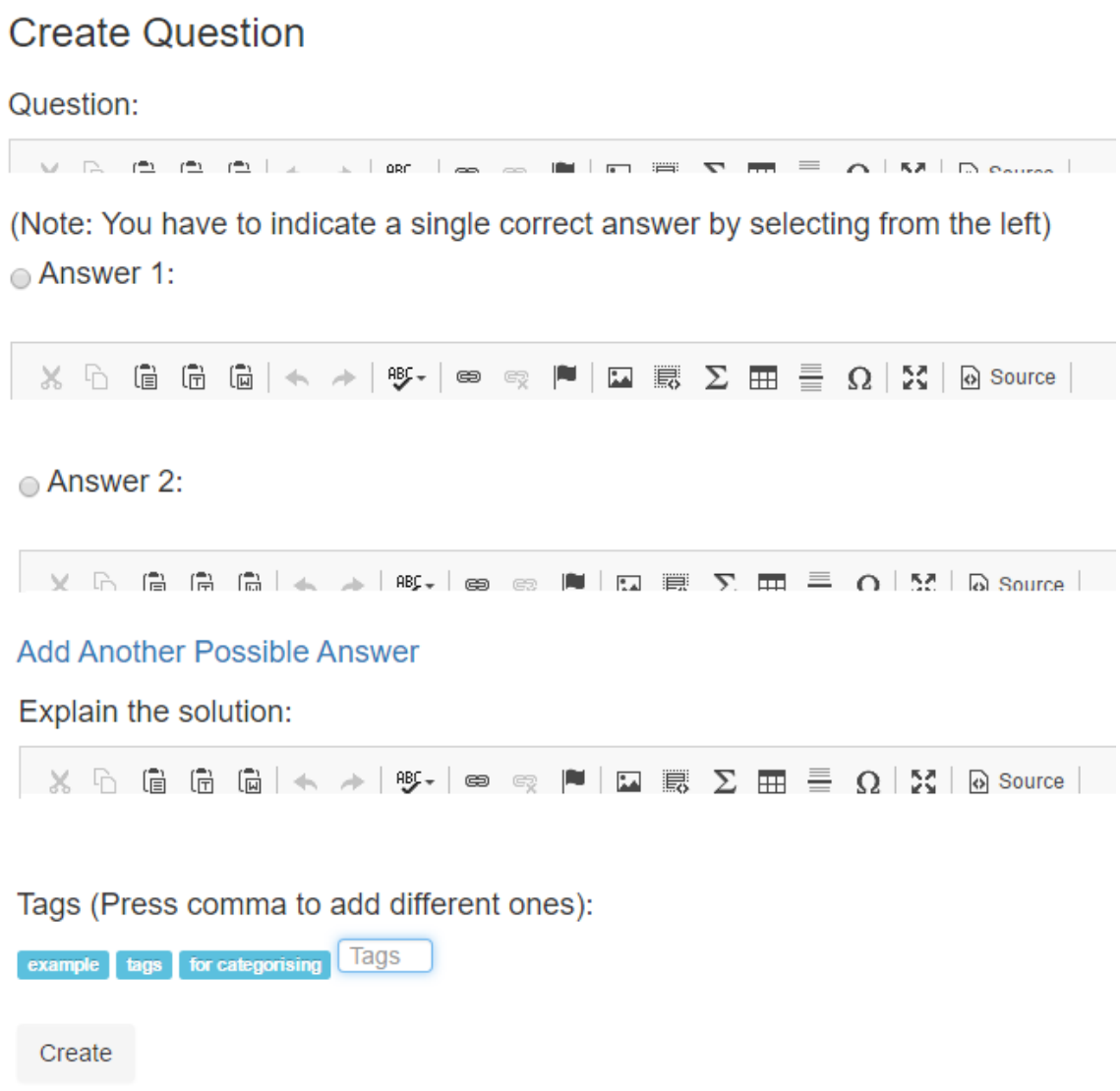


Figure 7 A cropped image of the form given to input questions and answers.

Once the question and the answers with the solution is put in, the user will be asked to put in some tags in order to help categorization. The tags will then be used as categories in the static home page.

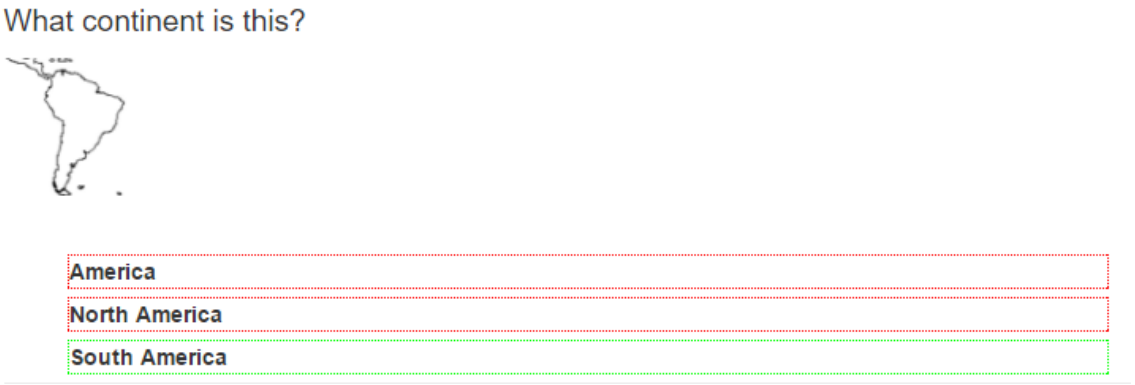


Figure 8 An example question and answer attempted by a user.

The person attempting to answer a question can than view the question with a multiple-choice answers. The user can go through all answers and click them. When an answer is wrong a red outline on the option is given to the user as shown in figure 8. This helps to allow the person to know what he or she has already attempted and that the answer was wrong.

3.6 Assignment Building

Sometimes it is important for a teacher to give out tasks as homework, one possibility is to create a group of questions in the form of an assignment for the teacher to give to his/her students. In most classrooms these are done in the form of assignments that are printed and handed out to the students. QuestionDB application tries to emulate this method of giving out assignments by creating a web based system with the same features.

Assignments are a great way for questions to get grouped together such that a student can study a topic specifically by doing an assignment on that topic.

4 Backend

The backend is the meat of the system. It will have to provide the storage and querying capabilities of the data.

While I attempt to adhere to the MVC (Model-View-Controller) architectural patterns for the system, the framework and third-party libraries I am currently using do not support it out of the box. However for clarity of communication I will try to define the business logic in terms of the different aspects of MVC components.

4.1 MongoDB

MongoDB is categorised as a NoSQL document database [1]. SQL database platforms store data as rows and columns in tables, with relationships between the table's columns. SQL is then used to query the data from the tables. NoSQL refers to alternative emerging technologies that store and retrieve data without the use of the row/column/table structures of data used in SQL. MongoDB is one of such systems. It is a document database that uses JSON (JavaScript Object Notation) to store documents through dynamic schemas.

The data management capabilities of MongoDB make it quite useful in certain applications and systems, providing better performance, or more rapid development abilities. For my system, the latter is one of the reasons why I have chosen it as the preferred data management solution. I have considerable experience of working with Node.js and MongoDB stack, so this makes them my preferred tool for the task at hand.

4.2 Nginx

While node.js is sufficient at running the business logic aspects of the application, it is however not a web server. Here Nginx comes to the rescue. With years of development gone into it for optimization as a web server, it can act as a battle-tested reverse proxy for my application server [6].

Configuring Nginx with a Node.js application is pretty straightforward. The Node.js application is ran alone (the application server) and Nginx can then be configured to relay requests it receives from the web to the Node.js application. Here Nginx acts as a reverse proxy. This allows for both systems to do what they are best at. Nginx adds its caching abilities, and fast static file hosting (such as client side JavaScript, CSS, images etc...) and Node.js is left to process the business logic of the site.

4.3 JavaScript and V8

JavaScript is an interpreted high-level programming language. It was originally developed for Netscape for its browser, to allow non-technical users to write simple scripts for creating interactive web pages. However it has grown well beyond its beginnings. Server side JavaScript was originally introduced by Netscape Enterprise Server. However it did not receive much attraction at the time due to many issues, such as the fact that the internet was still in its infancy and Java applets was the most active technology for client side application development. It has since outgrown these concerns.

With the advent of Google and its growth, Google has entered the browser market with the Chromium project. Google chrome was developed as an alternative to existing browsers, and with it V8 JavaScript engine was introduced as a lightweight and minimalistic JavaScript engine. V8 has since grown considerably, and a huge effort has been put into optimizing the engine for faster execution of JavaScript [7]. Due to its open source licensing, it has since been adopted by multiple open-source projects, Node.js, Couchbase, and MongoDB being some major projects relying on it.

The question still remains, why have I chosen JavaScript as the language for development on the server side? While there are many arguments as to what language or environment should be used for development, there is general consensus among developers as to the process in which an environment for development should be chosen [8]. The first step is to find an existing system that fulfils most of the requirements for the

system that needs to be developed, as there are no applications known to me that are readily available for me to store questions and give the user correct feedback regarding their answer choice.

I have to choose a battle tested web development framework for creating the CRUD (Create Read Update Delete) operations logic. There are many such frameworks, Ruby on Rails, Express.js, Django and CakePHP to name a few. The second step to this decision is, what toolkits I as the sole developer am most familiar and comfortable with. Django and Express.js are both frameworks I have worked with in the past, and consider them both equally capable of accomplishing the task at hand. However the web application may require quite a bit of frontend work with JavaScript, and switching between python and JavaScript (for development in Django) may not be quite. Thus I have decided to go with JavaScript being the main language for development.

4.3.1 Node.js

Node.js adds a lot of JavaScript libraries there were originally written in C/C++ for efficiency in order to create a full-fledged server side programming language out of JavaScript. As taken from the Node.js organization's website:

> Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world. [17]

The runtime is quite familiar to most developers, thanks to JavaScript. The second benefit that I think deserves a mention on its own is the fact that it is event driven. The event-driven non-blocking nature of it can make it extremely efficient in certain web architectures – most websites are event-driven by their nature, an event being that the user requests for a certain resource then the server responds with what it is programmed to respond with.

The npm (node package manager) apparently has grown to include so many libraries that it may try to contend with other more mature systems – such as Python's pip.

However it is arguable as to the quality of the packages compared to other environments which are much more mature. But maturity does not necessarily always mean quality. Systems built in more mature environments tend to use algorithms that are old but have to be left behind due to compatibility issues – the newer systems have more freedom to solve these drawbacks in order to create an overall better systems. However there is hardly anyone arguing the fact that older systems tend to be much more battle tested and production-ready.

However the adoption of Node.js by many corporations doing a production system seems to suggest that it's ready to handle certain tasks, and with more time it will only mature more.

4.3.2 MVC

MVC is a design pattern for software architecture. It was first conceived around 1970 in order to build GUI applications for smalltalk-76 (an early programming language attempting for a better human-computer interaction). It has since evolved further, and many variants of it have erupted - such as MVVM (Model View View Model), HMVC (Hierarchical MVC), etc... [9] MVC pattern was developed for creating graphic user interfaces on desktop computers, the intention being that if the application is divided into three different components, then the code would be easier to manage with clear responsibilities of each component outlined. It has now almost become the standard in web development. It allows the separation of the different parts and expertise required to develop a website. The separation is done by having three main components: the model, view and controller.

Explained in a simple way, the MVC components can be understood as each part being [10]:

- Model: the model is responsible for handling all data related requirements and behaviours. The model reacts to the application's demands for various information at different stages of the processing, it also makes updates and changes as is required on the data - and in an event driven system it will also send out alerts to responsible parts of the system to inform it as is required. The model is

the administration system of all the data handling requirements that an application requires. It can do this via various means, such as storing and retrieving information from the memory, or by delegating these requirements to a database and accessing the database in question with an abstraction layer. In my application this is done.

- View: The view defines the user interfaces, the means by which the end-user interacts with the system. The view of the MVC should define the way the UI is rendered in order to show the data that was got from the models. In a web application this usually means HTML/JavaScript/CSS that is written by the programmer. These assets are then rendered with the data gotten from the model by the controller to produce the visual websites seen by the visitor to the website.
- Controller: The controller holds the business/domain logic. The controller navigates the user, and decides what data to get from the model and then uses the data to render the right view. It can also use the proper views to get user input which it then redirects to the model so that the model can do updates on the data.

4.3.3 Controllers

CRUD (Create Read Update Delete) operations are necessary in order for the application to provide the information necessary for the end-user.

4.3.4 Views: Pug

In order to create the views I have used the templating engine called Pug (formerly known as Jade [11]). Pug is written in JavaScript and it compiles the Pug template source code that is given to it into a JavaScript function that takes a single data object called locals as its argument.

Here is a list of the pug source files and their main objectives in templating. These source files are located under the “views” directory.

layout.pug - All the other templates extend this template, and this can be considered the root of the entire website’s html. Content in here is shown on every page of the website. This template defines the html, head and body tags, along with many common JavaScript, CSS and image files.

This template also defines the menu that is shown, the menu items for the authenticated and also the guest users – the logic for which is passed into the template as the user object. The user object has a flag property “authenticated”, which when true means that it’s an authenticated user trying to view the web page.

The layout.pug file also implements a common JavaScript function that uses jQuery based notify.js system in order to give notifications to end-users from an Ajax-based request’s responses.

Finally, the layout.pug source code also contains multiple “block” that are utilised by the sub-views in order to add extra html source code into parts as required, them being:

- **additional_header** : this is in the “head” part of the html, this allows the sub templates to add extra html into the "head" element, such as extra JavaScript and CSS source.
- **content** : this is where the main content of the template go in, such as html form elements, or other elements that allows the end user to view something of interest - such as the questions or assignments list.
- **additional_body_end** : this block is at the end of the body element of the html, which is mostly useful for running JavaScript that one would like ran after the page is loaded.

question_form.pug - In order for the user to enter the question and the answers belonging to said question, the application renders this template to output the form elements required for user to view the form and enter the question and also add new answers and the answer value.

In order to do this, the template adds an additional part of JavaScript source to the additional_header block of the layout.pug template which it extends. Firstly a tagsinput.js file is added as a script element, this file loads up a JavaScript library that allows me to create a beautiful looking tag editing input system. The second part of JavaScript that is added is to allow the user to add extra options for answers should he/she will it, this is simply done by the script by appending an input and textarea element at the end of the element with the id attribute value of “answers”. The third part of the JavaScript ensures that an instance of CKEditor is loaded up, and applied to each textarea for question and answer input.

question_list.pug - To show the list of questions, and also to view assignments, this template is passed a list of question objects. The template than takes this list and loops through each object, than outputs the question in a div element, which contains further answer options which are put in as input and label element combinations.

When each of the label element is clicked (the label contains the answer option text), the “onclick” event is detected, the browser than runs the check_answer JavaScript function.

The function takes the label element as the function’s argument value, which his than used in order to get the element ID, which in turn allows us to know whether the answer is correct, and if the answer is correct a certain CSS class is added by the function to the answer – this allows for the website to give the user feedback regarding whether the answer selected was the correct answer or not.

The rest of this template consists mainly of CSS in order to properly stylize the different elements that exists in it.

Each question and answer is given the menu element options “Add For Assignment”, “Edit”, “Delete”. All of which requires the user to be authenticated in order to carry out the option successfully. However editing and deleting options are only given to the user, if the user has created the question himself/herself. Pagination data for this template is passed from the controller.

question_view.pug - This template is used for showing a single question, while ideally the question_list.pug file should include this file while looping through each question, the pug templating language does not allow for such feature and thus this file is solely used for showing a single question after an update or create operation takes place on a question.

question_tag.pug - This is a static template file which holds a lot of categories of questions, currently it is generated using static means because the amount of coding that might require in order to generate this file on the go each time is a considerable amount.

Seeing as I have to use multiple means to determine what are the most relevant tags of the question and how they define the question's category. Also an automatic system will also have to determine which tags are at which order – meaning some tags will have to sub categories of parents, and to further complicate matters this might nest into multiple levels.

assignment_list.pug - After an authenticated user have created assignments, the user might need to view the created list of assignments to copy the link for the assignment or to make updates to it. This template lists the assignments, and the links to view and edit the assignment.

assignment_view.pug – This is by far the simplest template currently, although in the future development will be required into it to make it more robust and add more features regarding assignment management. However currently it mainly renders question_list.pug with an extra part of code in order to show the name of the assignment (should the assignment be given a name.)

4.3.5 Models: Mongoose

The models are stored in MongoDB, which is a schema-less document database. However the idea of a schema provides a certain amount of clarity that is otherwise hard to achieve. In order to minimize the amount of code that I need to write in order to handle the data requirements of the model part of the code I have used the Mongoose library from the node package manager.

Mongoose can be considered to be a database abstraction layer. It uses schemas in order to provide certain features, e.g. the ability to validate certain fields before saving it to the database. Table 2 relates the different models that currently exists, and explains the types of data structures that are used to facilitate the ability for the application to store data.

Table 2. List of data model schemas created in mongoose.

Collections	**Description**
mcquestions	This schema holds data for the multiple-choice questions. **Fields**: question, solution, created_on, updated_on, votes_up, votes_down, tags (an array of string), created_by (a mongoose ObjectID), answers **Sub-fields**: answers is an array of sub documents which contain the answer,and a boolean flag field is_correct.
assignments	Holds the list of questions, and other meta data that is necessary in order to carry out assignments. Some of the features possible with these data structures are not implemented. **Fields**: name, created_by (ObjectId reference to user), questions (an array of ObjectId reference to question), created_on, password, name_required, tell_correct, tags (indexed string array), hours_duration
users	For user authentication the user data must be persisted, and this schema holds the data that is needed for login and also information about third party services – should the user use a third party service as an identity provider. **Fields:** email, password, name, facebook, google, login_at, register_at – all these fields are validated as string, except for the fields ending with "_at", these fields are stored as date data. **Sub-documents:** for facebook and google field, they contain sub documents which contain id, email, token and name. These data is all provided by the identity providers respectively, during

	registration of the user my application receives a redirect from the identity provider with details that the OAuth can use to query for this information.
assignmentresults	Is not currently implemented feature wise. However the aim is to allow users to do assignments online and store results for the assignment maker to view.
questiondifficulty	Is also not implemented. The collection stores how many users have attempted the question and how many got it right. This measure along with the votes for the question allows for sorting questions in terms of quality.

4.4 Authentication and Single Sign On

Authentication for the system is required as there is a need for knowing who created each question to decide who can edit it. The same also applies to assignments. There are multiple aspects to proper authentication on a web application, a user must first be authenticated based on a username/email and password combination. The storage of this information must be encrypted and kept secured. The emails are also very useful in order to contact the user for various reasons for the application. However there is always the chance that the email might be invalid and not verified. In order to minimize all these aspects of user verification and the system that goes behind it, I have decided to allow authentication via oauth, which is commonly also referred to as a single sign-on process.

The following are the steps that are taken for a 3rd-party based authentication using SSO [12]:

- The user wants to authenticate for the website in order to have access to certain resources of the website. For QuestionDB the user will need to authenticate in order to be able to create questions and handle assignments.
- The system realises that the user is not authenticated, and that he/she wants to authenticate to the system. So the system (passport.js library in my case) sends a redirection to the user with a url that is encoded with information for the identity provider.
- The end-user than gets its browser redirected to the identity provide with application specific information – such as app ID for Facebook SSO.
- If the user is already authenticated by the identity provider, than the user is asked usually for giving consent to the application, otherwise the user is asked for his/her login details (such as username and password).
- As soon as the identity provider has authenticated the user, and gotten his permission to authenticate for the application, the identity provider than creates a response that needs to be given, it than gives the end-user this information along with a redirect url back to my application.
- Once the user's browser receives the redirect url and other information, it redirects the user to my application to a callback URL. In my system I have configured the url "/auth/facebook/callback" as the callback url.
- After the identity provider has authenticated and redirected, the application is given an access token which my application can then use in order to get more information about the user – such as full name, email address, etc… The scope of information that the system need to access

OAuth is an industry standard to allow users to authenticate themselves with ease [13]. Based on the HTTP protocol, it has become a very popular method for many internet users/developers to login and authenicate themselves with. Basically a third party such as Google, Facebook, Microsoft Is used as the identity provider and the developer for the system then uses this identity in order to authenticate the user. It allows for the developer to not have to write a system for all the aspects of authentication, and it also has the added benefit of allow the user to not have to share their password with us, this also means that the end user has to put in less work.

Oauth is a standardized process by which the authentication flow can be done without the end-user having to share their login details. Ouath is based on a system whereby the resource manager (my application) sends a request to a third party identity provider (such as Facebook, Google, etc...) to get an authentication token, if the user agrees with the identity provider to share his authentication for the resource manager, than the identity provider calls back a URL in the application with an authentication token that is provided to the application.

The application can use this authentication token in order to get further information from the identity provider regarding the user's identity, such as email, name, address, depending on what request the system sent to the identity provider, it will than ask the user if they are willing to share that information and the application will then be able to use the access token provided in order to get the information the application have been granted permission for.

For easier implementation, I have used passport.js library from the node package manager (npm for short). Passport.js handles the nitty gritty parts of authentication, for instance creating a session with the proper data in it,

5 Discussion

5.1 Future Tasks

There is much to be done in order to truly create the system that I aimed for. While creating multiple-choice questions, and giving end-users some feedback regarding the accuracy of the answer choice that they made is a first step, it is by no means complete and a final product. Table 3 tries to build a comprehensive list of all the tasks that I would like to have done in order to truly have a question database system, to my desire, built:

Table 3. Suggested improvements

Task	**Description**
Fill in the blank	Create question system where the user can fill in the blank
Regex based answers	This is for questions which accept answers that are corrected based on whether the answer matches certain regex expressions – this might be useful for getting answers with proper use of units for instance in physics. However the person creating the questions must be somewhat familiar with regex systems.
Points based system	Have a better permissions system, where by each user is given points based on how much questions they input
Question difficulty measure	In order to measure the difficulty of the questions, there can be two measures. One measure of difficulty can be provided by the person publishing the question. The second measure (perhaps more reliable) is by detecting the number of persons who got the answer wrong in the first try from all the people who have attempted the question. However the second measure can not be very reliable unless the question is attempted by a significant number of users. Knowing the difficulty of a question for a user is important because that allows educators to get an insight as to what type of problems are the students struggling currently with, or in general what topics do students seem to struggle in – this will allow the teacher to more effectively use his/her time educating the students.
Better analytics	Analytics can be applied to many parts of the system, such as collecting more data as to how long a student stays in a question, how many tries does it take the student in order to get the right answer on a question, what types of questions tends to lead to the students moving to harder questions more effectively, the list goes on.
Exporting assignments	The aim of the system that I wanted to create was not just for the students to practice questions but I wanted to use it in order to make the teacher's life easier also. My hope was to allow the teacher to export the assignments which they can print into physical copies, for students to practice in class rooms. The teacher can get an endless number of questions from other teachers who contribute

	to the database, this leads to all educators benefitting from creating more and more unique ways of asking questions to really see what a student's mind is thinking regarding the material he or she is studying.
Online assignment and grading	Perhaps another possibility of this system is to use it in order to let the users create assignments that they can use for grading. By not showing the end user the correct answer, and also limiting the time that an assignment is open to a student, I can create a system that be used for quizzing the students. This gives the students the flexibility to take a quiz at their own leisure. And frees up the teacher from the menial task of grading papers.

5.2 Testing

We have to test the software that was developed in order to give an objective idea to the stakeholder regarding how the software reacts under different conditions. The intention of software testing is basically to ensure quality control of the system that is developed. In this way we can get the software that best serves the needs it was developed for and reacts in the most predictable way possible.

5.2.1 Beta Testing

Beta testing is a type of user acceptance testing. Before a software is to be released for the use by the end-user, the system will be released to a few selected users in order to see how those users react to the functionality of features of the software system in question [14]. As beta testing is not a system testing, where one checks the system so that it doesn't crash or fail in anyway in required system states of the system. It is used mainly to see that the system actually solves the problem it was developed for.

Beta testing is perhaps the most common type of testing that is regularly applied to modern web applications that are developed. Due to the ease at which it can be carried out and the results that can be obtained from such tests.

5.3 Scalability

Scalability is hard to define simply. A system is believed to be scalable when it can be used to solve the same problem at a larger amount. A scalable system is one which is efficient and feasible to run and process data at a larger amount continuously, this can be to support a large amount of users concurrently, or a large amount of output and so on.

As an example, say there is a chat application that is written to support 10 concurrent users by a single node (let us say a node in this example is a server in a cluster of servers.) Then if for 20 users it requires 5 nodes, then this system is not very scalable (double the number of users, but five times as much resource needed). Such issues can be a big problem from a business perspective - as this increases costs considerably without giving as much output as it should. However, using the same system as an example, if now 100 users can be supported with only 2 nodes (one extra node, 10 times the user), then the system is said to scale well.

Scalability is needed for any viable software systems in order to make sure that should the system grow one can "scale" it to allow it to support a larger amount of data and users.

5.4 Security

While security is not of the utmost concern in this web application, a basic standard of it must be applied to any system, in order to maintain a level usability, control and trust in it. Writing about the many flaws and discussing the possibilities of security threats that might be faced by a web application is a task that is best left to a book. However I will

consider some of the measures I have taken in order to minimize the possibility of an intrusion:

- SSH Login is only allowed from a certain IP range, this ensures that an attacker will have to gain access to my VPN before being able to use SSH to login and do updates and changes to the server.
- The Node.js application that runs and serves the application is behind nginx, which is a known and battle tested http server. This means that nginx is truly what is facing the public for the website.
- In order to worry less about keeping the database content encrypted and secure at all times, I have not stored any sensitive information in the database other than very basic user information such as email, name and access token for oauth service providers. Theoretically an attacker can gain access to the email and access tokens, but the oauth permissions that my application requires is extremely minimal – and thus the effects are minimized. However the email addresses can be abused for spam or scamming purposes – but this is best left alone for email filters.
- No sensitive data is shared on the client side.
- No data received from the client side is used directly for querying the database, the information is encoded than run in the database.

5.5 Limitations

Multiple limitations on the existing system still exists. The system is currently limited by many aspects, which are mainly issues with the existing features:

- Static home page : The current home page is static. I have hard coded the tags as categories. The reason behind this is that determining the correct tags to automatically show in the home page requires an algorithm that would require it's own thesis to be written on. The categories must be related and further combined. It will also require a significant input of questions and answers with their tags in order to calculate what tags to show together, and under what tags. In order to solve this problem I believe a cron job has to be ran which goes through all the tags data and generates an index out of it. This information can

then be used in a word cloud in order to show what types of questions currently exist the most in the website.

- Multiple-choice questions: The current system only allows for multiple-choice questions. While multiple-choice questions are quite good, they may not always be the most suitable way a person wants to practice. Some questions are better off as fill in the blanks, or just open ended text answers. While the computer will probably not be able to correct essay answers, there are other types of questions that I would like implemented, fill in the blanks and multi select are ones that come to me immediately.
- Single sign on with multiple IDP results in multiple accounts, regardless of email address. Currently I do not have the implementation in order to check if the email address that I got from an IDP already exists in the database or not, and if it exists the user should be logged in with that account. While this maybe an issue currently, some users may actually use it as a feature in order to create multiple accounts – for their own reasons, which is very common for a lot of online users.
- MathJax is client side rendering: While the client side ensures that the backend has to do less work, it also means that my compatibility is limited since MathJax is somewhat modern and uses many features not supported by (considerably) older browsers and lower power devices. In order to solve this, and also to get a consistent equation, it might make more sense to render it in the server side. Server side rendering will also have the added benefit of allowing to get an image that I can use in order to allow for users to export assignments as documents.
- Correct answer detection: Currently the correct answer is detected and shown with a mix of CSS and JavaScript, both of which are client side. For now, it is fine since I only want the user to practice questions. But in the future when I would like to allow teachers to use this system to make quizzes, it would be important that the client does not know the correct answers, and answer correction is done on the server side.
- Local user database with password based authentication: currently authentication is mainly done via 3rd party identity providers. It would make sense to allow for users to register using a form and then login with another form, this is an op-

tion that is always a good idea to provide to users, as some users may not be comfortable with connecting their third party accounts to my site.

5.5.1 Mistakes and Errors

Any system that is developed is bound to have mistakes. Only by looking back at it and finding them can one improve and learn from their mistakes. So this section will discuss the mistakes and regrets that I have as the architect of this system and what I plan on doing in order to try to improve myself in the future and not repeat these mistakes.

The system that I created, while tries to separate the business, data and view logic from each other (to be truly MVC). I am afraid that my code is not truly as MVC as I would like it to be. For instance a lot of the controller code actually contains code that should be separated to the model. Some of the code unintentionally has been mixed up in the different parts where it should have been separated, a better way to get about it and fix the situation would be perhaps to redesign the system on a different framework, due to other mistakes that have potentially been done.

MongoDB as the data storage engine was chosen due to its ease of data handling, and the ability to handle document data types. MongoDB is schema-less and this makes it ideal since in the future I could easily switch the data types. However there are many scenarios where a more traditional SQL DBMS would outperform the current setup. One such scenario is for instance the User models, and the related data. However the question to use a two-database system may not be the best as this adds over-head of maintaining two different systems. This means that there are two options, switch the database to MySQL or leave it with the current system.

5.5.2 **Benchmarks**

A common way to check how well a system performs under a production environment is by running benchmarks on it. The Apache foundation provides a system for this

called the Apache Benchmark (ab command in most linux distro where it is installed [15].)

The ab system takes in the argument for a number of concurrent users and a number of requests to do and the target URL to test against, as an example the command in listing 3 will benchmark mypage URL with 100 requests using 10 concurrent users, meaning 10 requests for 10 different users at the same time:

```
ab -n [number of request] -c [number of consecutive connections] [target url]
ab -n 100 -c 10 http://localhost/mypage
```

Listing 3. apache benchmarking syntax.

Having done the benchmarking (more thorough results of it can be found in Appendix III), there seems to be a considerable issue in the benchmark results that show weakness for the system. The reason I think that the choice of node.js framework was a bad decision, performance tuning and fixing is a huge task, and could be left to a thesis of its own. So I will for the time being just discuss possible pitfalls that can be seen from the benchmarking results.

To start with, if we look at a static file served by node.js for 100 requests, with 10 concurrent user, the result is:

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:     4    5   1.3      5       9
Waiting:        3    4   1.1      4       7
Total:          5    5   1.3      5       9
```

Listing 4. ab output for 100 request, 10 concurrent users

As expected the result is nothing out of the ordinary, however with a few more users of 100 concurrent users, and 1000 requests:

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.9      0       4
Processing:    24   55  15.0     48      88
Waiting:       16   45  15.2     39      80
Total:         28   55  14.8     49      88
WARNING: The median and mean for the initial connection time are not
within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     49
  66%     51
  75%     51
  80%     81
  90%     86
  95%     87
  98%     87
  99%     88
 100%     88 (longest request)
```

Listing 5. ab output for 1000 request, 100 concurrent users

This shows a somewhat linear performance with higher loads. However to try and break the system I have tried to go higher with 10000 requests, and 1000 concurrent users, which lead to fall in the performance a significant number of requests were taking well acceptable response times:

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  263 694.6      1    3006
Processing:    60  226 289.9    197    6421
Waiting:       44  187 287.8    165    6419
Total:         62  489 796.9    223    7425

Percentage of the requests served within a certain time (ms)
  50%    223
  66%    242
  75%    392
  80%    405
  90%   1195
  95%   3162
  98%   3410
  99%   3411
 100%   7425 (longest request)
```

Listing 6. ab output for 10000 request, 1000 concurrent users

Now we can see that the server is starting to reach its limit, with about 10% of the requests taking more than a second. The above results were only done for static files, which tend to be served pretty fast since there is very little work to be done on reading the file over to the network. If I take a very simple dynamic page generated by node.js, we will see how badly performance starts to fail. Requesting the home page directly with 10 concurrent users for 100 requests will lead to the following result:

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:   281 1081 455.1    975    2802
Waiting:       94  473 432.9    369    2148
Total:        281 1081 455.2    975    2802

Percentage of the requests served within a certain time (ms)
  50%    975
  66%    977
  75%    985
  80%    985
  90%   2151
  95%   2431
  98%   2617
  99%   2802
 100%   2802 (longest request)¨
```

Listing 7. ab output for 100 request, 10 concurrent users for dynamic page.

This is a very poor performance considering that so few request need to be made with so little concurrent users, and why I think node.js requires performance tuning for the application. Since I believe I have certain parts of the code that are blocked – a possible solution for this can be caching. I am also thinking of changing the templating engine to something simpler and faster.

6 Conclusion

The aim of the project was to create a web application that can store the questions and answers, and allow the end user to enter new pairs of questions and answers. Other users should then be able to use all the data entered in order to practice and get immediate feedback regarding their input.

In conclusion the application does provide the minimal features required in order for it to function well to serve the needs of the users to enter questions and answers, and to use them in order to practice for topics that they might have difficulty in understanding.

While the choice of node.js was not the best choice due to the performance that it gives in this application, the application is still functioning but not scaling well. This means that performance tuning is vital to widespread use of the system. The biggest mistake was perhaps using a framework meant for real-time web systems design in a web application that is somewhat more like a traditional website.

The database choice of MongoDB is one I still stand by due to the fact that it allows for schemaless data storage, meaning that new data structures will be easy to implement should I ever decide to move away from simply having multiple-choice questions to other types of questions – such as fill in blanks, essay type of answer and others.

The project is successful from the perspective that a usable application has been developed and while it can be considered as a design developed for demonstration purposes, it is far from being production ready, at least not without a lot of performance gains.

References

1. Ivenza, DTAP for releasing new software Versions [ONLINE] Available at: http://www.ivenza.com/blog/dtap-for-releasing-new-software-versions/ [Accessed 28 February 2017]

2. Git - git-svn Documentation. 2017. Git - git-svn Documentation. [ONLINE] Available at: https://git-scm.com/docs/git-svn. [Accessed 28 February 2017].

3. Cloud9. 2017. Cloud9 - About Cloud9. [ONLINE] Available at: https://c9.io/site/about. [Accessed 28 February 2017].

4. CUergo: Ergonomic Guidelines for Interface Design. 2017. CUergo: Ergonomic Guidelines for Interface Design. [ONLINE] Available at: http://ergo.human.cornell.edu/ahtutorials/interface.html. [Accessed 28 February 2017].

5. Introduction to LaTeX. 2017. Introduction to LaTeX. [ONLINE] Available at: http://www.latex-project.org/about/. [Accessed 28 February 2017].

6. nginx. 2016. nginx. [ONLINE] Available at: https://nginx.org/en/. [Accessed 13 December 2016].

7. Google Developers. 2017. Chrome V8 | Google Developers. [ONLINE] Available at: https://developers.google.com/v8/. [Accessed 28 February 2017].

8. Computing Australia. 2017. Software Development – when to go the custom route and what to expect | Computing Australia. [ONLINE] Available at: https://computingaustralia.com.au/software-development-when-to-go-the-custom-route-and-what-to-expect/. [Accessed 28 February 2017].

9. Niraj Bhatt - Architect's Blog. 2017. MVC vs. MVP vs. MVVM | Niraj Bhatt - Architect's Blog. [ONLINE] Available at: https://nirajrules.wordpress.com/2009/07/18/mvc-vs-mvp-vs-mvvm/. [Accessed 28 February 2017].

10. Tom Dalling. 2017. Model View Controller Explained. [ONLINE] Available at: http://www.tomdalling.com/blog/software-design/model-view-controller-explained/. [Accessed 28 February 2017].

11. Getting Started – Pug. 2017. Getting Started – Pug. [ONLINE] Available at: https://pugjs.org/api/getting-started.html. [Accessed 28 February 2017].

12. Danial Khosravi. 2016. Authentication Using PassportJS - Danial Khosravi's Blog. [ONLINE] Available at: http://danialk.github.io/blog/2013/02/23/authentication-using-passportjs/. [Accessed 13 December 2016].

13. OAuth Community Site, 2017. [ONLINE] Available at: https://oauth.net/. [Accessed 28 February 2017].

14. Centercode | Beta Test Management Software and Managed Betas. 2017. Alpha vs. Beta Testing | Centercode Blog. [ONLINE] Available at:

https://www.centercode.com/blog/2011/01/alpha-vs-beta-testing/. [Accessed 28 February 2017].

15. ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4 [ONLINE] Available at: http://httpd.apache.org/docs/2.4/programs/ab.html. [Accessed 13 December 2016].

16. MongoDB Documentation. 2016. MongoDB Documentation. [ONLINE] Available at: https://docs.mongodb.com/. [Accessed 13 December 2016].

17. Node.js Foundation. 2016. Node.js. [ONLINE] Available at: https://nodejs.org/en/. [Accessed 13 December 2016].

18. Mongoose ODM v4.7.2. 2016. Mongoose ODM v4.7.2. [ONLINE] Available at: http://mongoosejs.com/. [Accessed 13 December 2016].

19. Brad Dayley, 2014. Node.js, MongoDB, and AngularJS Web Development (Developer's Library). 1 Edition. Addison-Wesley Professional.

20. DigitalOcean. 2016. How To Set Up a Node.js Application for Production on Ubuntu 14.04 [ONLINE] Available at: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-14-04. [Accessed 13 December 2016].

21. Express - Node.js web application framework. 2016. Express - Node.js web application framework. [ONLINE] Available at: http://expressjs.com/. [Accessed 13 December 2016].

22. Gist. 2016. Default Nginx Conf · GitHub. [ONLINE] Available at: https://gist.github.com/nishantmodak/d08aae033775cb1a0f8a. [Accessed 13 December 2016].

23. Google Developers. 2016. Chrome V8 | Google Developers. [ONLINE] Available at: https://developers.google.com/v8/. [Accessed 13 December 2016].

24. David Herron, 2016. Node.JS Web Development - Third Edition. 3rd Revised edition Edition. Packt Publishing - ebooks Account.

25. Eelco Plugge, 2010. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing (Expert's Voice in Open Source). 1st ed. Edition. Apress.

26. Passport.js. [ONLINE] Available at: http://passportjs.org/. [Accessed 13 December 2016].

Appendix I : package.json

This file defines the system that I current have and the dependencies that are required by the system.

```
{
  "name": "questiondb",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "start": "node app.standalone.js"
  },
  "dependencies": {
    "express": "3.2.6",
    "pug": "2.0.0-beta6",
    "mongoose":"4.1.0",
    "passport": "0.3.0",
    "passport-local": "1.0.0",
    "passport-facebook":"*",
    "passport-google-oauth":"*",

    "multer":"*",

    "cookie-parser": "*",
    "body-parser":"*",
    "method-override":"*",
    "express-session":"*"

  }
}
```

Appendix II: models.js

Model file content for creating ”schemas” for mongoose to use on mongodb are as follows:

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema;

/* schemas */

/*
mongoose.model('User',{
        username: String,
    password: String,
    email: String,
    gender: String,
    address: String
});
*/

var userSchema = new Schema({
  email: String, // todo: add local authentication.
  password: String,
  name: String,
  facebook: {
    id : String,
    email: String,
    token: String,
    name: String,
  },
  google : {
    id : String,
    email: String,
    token: String,
    name: String,
  },
  login_at: Date,
  register_at : {type: Date, default:Date.now}
})

var questionSchema = new Schema({
    question: String,
    created_by :  mongoose.Schema.Types.ObjectId,
    solution: String,
    create_on: {type: Date, default: Date.now },
    updated_on: {type: Date, default: Date.now},
    votes_up: Number,
    votes_down: Number,
    tags: [String],
    answers: [{
      answer: {type: String},
      is_correct: {type: Boolean, default: false }
    }],
})
var tagSchema = new Schema ({
    questions: [{ type: Schema.Types.ObjectId, ref: 'MCQuestion' }],
})
var assignmentSchema = new Schema ({
    name: {type:String, default:"untitled"},
    created_by :  mongoose.Schema.Types.ObjectId,
    questions: [{ type: Schema.Types.ObjectId, ref: 'MCQuestion' }],
    created_on : {type: Date, default: Date.now},
    password: { type: String, default: null },
    name_required: Boolean,
    tell_correct: Boolean,
    tags: {type: [String], index: true},
    hours_duration: Number
```

```
})

var assignmentResult = new Schema ({
  answer_by: {type:  mongoose.Schema.Types.ObjectId, ref: 'User'},
  assignment_start : Date,
  assignment: {type:  mongoose.Schema.Types.ObjectId, ref: 'Assign-
ment'},
  answers: [{
    question: { type: Schema.Types.ObjectId, ref: 'MCQuestion'},
    is_correct: Boolean
    }] ,
  answer_number: Number ,
  client_info : String
})

//newAve = ((oldAve*oldNumPoints) + x)/(oldNumPoints+1)
var questionDifficultySchema = new Schema({
  for_mcquestion: {type: Schema.Types.ObjectId, ref: 'MCQuestion'},
  current_average_tries: Number,
  number_of_tries_count: Number,
})

var tMCQuestion = mongoose.model('MCQuestion', questionSchema);
var tAssignment = mongoose.model('Assignment', assignmentSchema);
//var tTag = mongoose.model('QTag', tagSchema);
var tQuestionDifficulty = mongoose.model ('QuestionDifficulty', ques-
tionDifficultySchema)
var tUser = mongoose.model("User", userSchema)

/* validations */


tMCQuestion.schema.path('question').validate(function (value) {
//validate question
  return (value.length > 0 ? true : false );
}, 'Question can\'t be empty.');

tMCQuestion.schema.path('answers').validate(function (value) { //
  return (value.length > 1 ? true : false );
}, 'Question must have a minimum of 2 answers.');

tMCQuestion.schema.path('tags').validate(function (value) { //
  var retval = false;
  value.forEach(function(v){
    v = v.replace(/\s/g, '') ;
    if (typeof v != "undefined" && v!= "" && v!= null){
      retval = true;
    }
  })
  return retval;
}, 'Atleast 1 tag must be provided.');

tMCQuestion.schema.path('answers').validate(function (value) { //
  var rt = false;
  value.forEach(function(ans){
      if(ans.is_correct){ rt = true};
  });
  return rt;
}, 'Select the correct answer.');
```

Appendix III: Benchmark Output

In order to generate the benchmark results, the following script was used:

```
echo ==================== >> result.txt
ab -n 100 -c 10  http://127.0.0.1:3000/javascripts/qdb.js >> re-
sult.txt
echo ==================== >> result.txt
ab -n 1000 -c 100  http://127.0.0.1:3000/javascripts/qdb.js >> re-
sult.txt
echo ==================== >> result.txt
ab -n 10000 -c 1000  http://127.0.0.1:3000/javascripts/qdb.js >> re-
sult.txt
echo ==================== >> result.txt
ab -n 100000 -c 10000  http://127.0.0.1:3000/javascripts/qdb.js >> re-
sult.txt
echo ==================== >> result.txt
ab -n 100 -c 10  http://127.0.0.1:3000/ >> result.txt
echo ==================== >> result.txt
ab -n 1000 -c 100  http://127.0.0.1:3000/ >> result.txt
echo ==================== >> result.txt
ab -n 10000 -c 1000  http://qdb.mursalat.net/javascript/qdb.js >> re-
sult.txt
```

Which resulted in the following content of result.txt:

```
====================
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient).....done


Server Software:
Server Hostname:        127.0.0.1
Server Port:            3000

Document Path:          /javascripts/qdb.js
Document Length:        1248 bytes

Concurrency Level:      10
Time taken for tests:   0.055 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      151300 bytes
HTML transferred:       124800 bytes
Requests per second:    1810.48 [#/sec] (mean)
Time per request:       5.523 [ms] (mean)
Time per request:       0.552 [ms] (mean, across all concurrent re-
quests)
Transfer rate:          2675.05 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:     4    5   1.3      5       9
Waiting:        3    4   1.1      4       7
Total:          5    5   1.3      5       9

Percentage of the requests served within a certain time (ms)
  50%      5
```

```
  66%      5
  75%      6
  80%      6
  90%      9
  95%      9
  98%      9
  99%      9
 100%      9 (longest request)
==========
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)


Server Software:
Server Hostname:        127.0.0.1
Server Port:            3000

Document Path:          /javascripts/qdb.js
Document Length:        1248 bytes

Concurrency Level:      100
Time taken for tests:   0.562 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      1513000 bytes
HTML transferred:       1248000 bytes
Requests per second:    1780.01 [#/sec] (mean)
Time per request:       56.180 [ms] (mean)
Time per request:       0.562 [ms] (mean, across all concurrent re-
quests)
Transfer rate:          2630.03 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.9      0       4
Processing:    24   55  15.0     48      88
Waiting:       16   45  15.2     39      80
Total:         28   55  14.8     49      88
WARNING: The median and mean for the initial connection time are not
within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     49
  66%     51
  75%     51
  80%     81
  90%     86
  95%     87
  98%     87
  99%     88
 100%     88 (longest request)
==========
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)


Server Software:
Server Hostname:        127.0.0.1
Server Port:            3000

Document Path:          /javascripts/qdb.js
```

```
Document Length:        1248 bytes

Concurrency Level:      1000
Time taken for tests:   7.460 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      15130000 bytes
HTML transferred:       12480000 bytes
Requests per second:    1340.55 [#/sec] (mean)
Time per request:       745.962 [ms] (mean)
Time per request:       0.746 [ms] (mean, across all concurrent re-
quests)
Transfer rate:          1980.72 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  263 694.6      1    3006
Processing:    60  226 289.9    197    6421
Waiting:       44  187 287.8    165    6419
Total:         62  489 796.9    223    7425

Percentage of the requests served within a certain time (ms)
  50%    223
  66%    242
  75%    392
  80%    405
  90%   1195
  95%   3162
  98%   3410
  99%   3411
 100%   7425 (longest request)
====================
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
====================
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient).....done


Server Software:
Server Hostname:        127.0.0.1
Server Port:            3000

Document Path:          /
Document Length:        4389 bytes

Concurrency Level:      10
Time taken for tests:   10.862 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      470490 bytes
HTML transferred:       438900 bytes
Requests per second:    9.21 [#/sec] (mean)
Time per request:       1086.201 [ms] (mean)
Time per request:       108.620 [ms] (mean, across all concurrent re-
quests)
Transfer rate:          42.30 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:   281 1081 455.1    975    2802
```

```
Waiting:       94  473 432.9    369    2148
Total:        281 1081 455.2    975    2802

Percentage of the requests served within a certain time (ms)
  50%    975
  66%    977
  75%    985
  80%    985
  90%   2151
  95%   2431
  98%   2617
  99%   2802
 100%   2802 (longest request)
====================
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)


Server Software:
Server Hostname:        127.0.0.1
Server Port:            3000

Document Path:          /
Document Length:        4389 bytes

Concurrency Level:      100
Time taken for tests:   97.050 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      4704668 bytes
HTML transferred:       4389000 bytes
Requests per second:    10.30 [#/sec] (mean)
Time per request:       9705.039 [ms] (mean)
Time per request:       97.050 [ms] (mean, across all concurrent re-
quests)
Transfer rate:          47.34 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.9      0       4
Processing:   387 9704 2418.4   9717   18855
Waiting:       97  803 1647.7    370    9622
Total:        387 9704 2418.9   9717   18857

Percentage of the requests served within a certain time (ms)
  50%   9717
  66%   9756
  75%   9809
  80%   9829
  90%   9941
  95%  14406
  98%  17228
  99%  18070
 100%  18857 (longest request)
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Total of 8578 requests completed
```

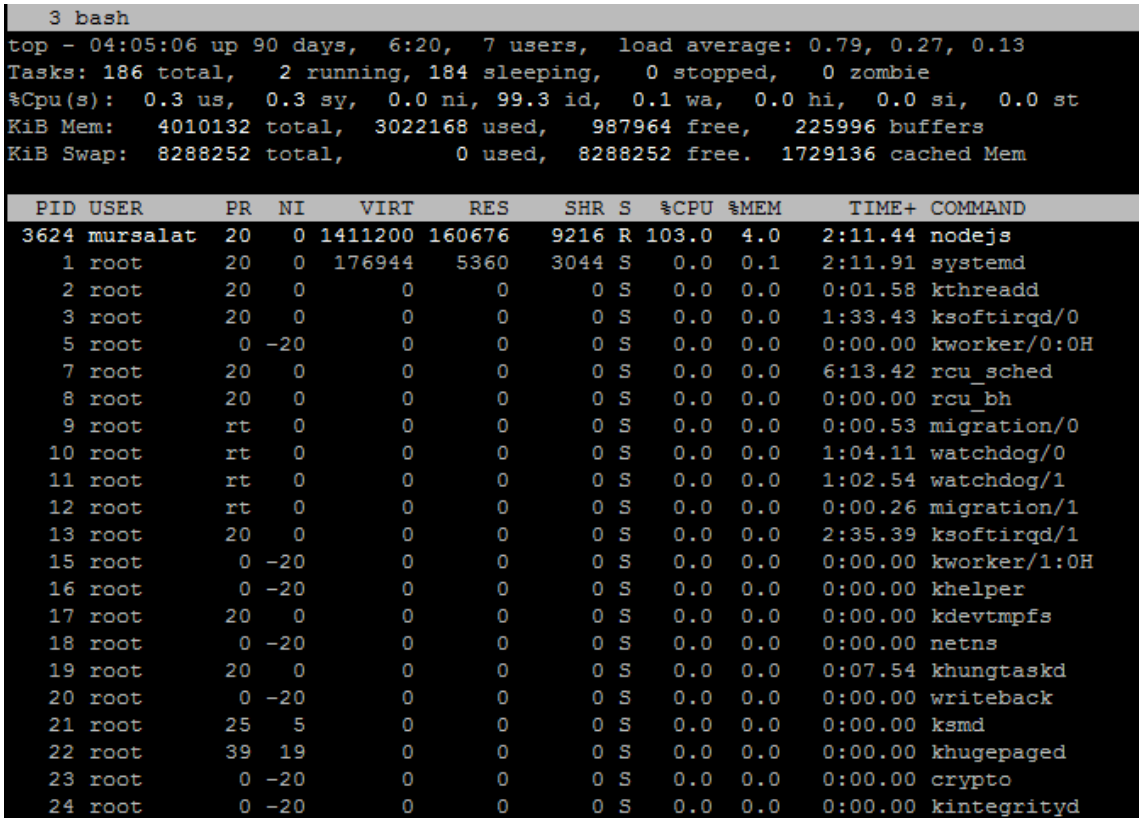
```
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
```

Benchmarking qdb.mursalat.com (be patient)

```
Server Software:        nginx/1.10.1
Server Hostname:        qdb.mursalat.com
Server Port:            80

Document Path:          /javascripts/qdb.js
Document Length:        193 bytes

Concurrency Level:      1000
Time taken for tests:   1.931 seconds
Complete requests:      10000
Failed requests:        9006
   (Connect: 0, Receive: 0, Length: 9006, Exceptions: 0)
Non-2xx responses:      994
Total transferred:      1094014492 bytes
HTML transferred:       1091719042 bytes
Requests per second:    5179.25 [#/sec] (mean)
Time per request:       193.078 [ms] (mean)
Time per request:       0.193 [ms] (mean, across all concurrent re-
quests)
Transfer rate:          553337.50 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   56 216.2      4    1009
Processing:    12   54  92.5     38     817
Waiting:        2   23  96.8      5     817
Total:         14  110 267.1     42    1820

Percentage of the requests served within a certain time (ms)
  50%     42
  66%     42
  75%     53
  80%     66
  90%     74
  95%    518
  98%   1082
  99%   1088
 100%   1820 (longest request)
```

Appendix IV: Process Use During Benchmark

The following is a screenshot of the output from top in linux, which shows resource consuptions by the process. Nodejs process seems to eat up CPU usage, and this is why a lot of the benchmark is very slow.



```
   3 bash
top - 04:05:06 up 90 days,  6:20,  7 users,  load average: 0.79, 0.27, 0.13
Tasks: 186 total,   2 running, 184 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   4010132 total,  3022168 used,   987964 free,   225996 buffers
KiB Swap:  8288252 total,        0 used,  8288252 free.  1729136 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3624 mursalat  20   0 1411200 160676   9216 R 103.0  4.0   2:11.44 nodejs
    1 root      20   0  176944   5360   3044 S   0.0  0.1   2:11.91 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:01.58 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   1:33.43 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   6:13.42 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.53 migration/0
   10 root      rt   0       0      0      0 S   0.0  0.0   1:04.11 watchdog/0
   11 root      rt   0       0      0      0 S   0.0  0.0   1:02.54 watchdog/1
   12 root      rt   0       0      0      0 S   0.0  0.0   0:00.26 migration/1
   13 root      20   0       0      0      0 S   0.0  0.0   2:35.39 ksoftirqd/1
   15 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   16 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 khelper
   17 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   18 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 netns
   19 root      20   0       0      0      0 S   0.0  0.0   0:07.54 khungtaskd
   20 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 writeback
   21 root      25   5       0      0      0 S   0.0  0.0   0:00.00 ksmd
   22 root      39  19       0      0      0 S   0.0  0.0   0:00.00 khugepaged
   23 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 crypto
   24 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kintegrityd
```

Appendix V: Terms and Abbreviations

Regex: Regular expressions, these are strings that represent patterns that can be matched to other strings.

HTTP: Hyper Text Transfer Protocol is a standard for communicating extensible markup language documents over the web to show in browser, should this end in “S” as in HTTPS, it means that it is secure with an encryption on top of it.

SSH: Secure shell, a secure and encrypted method to communicate over the network to manage and control services.

SVN: Apache Subversion is a version control system.

CRUD: Create Read Update Delete are operations that are usually always carried out in systems where data is stored in a database.